Specification Amendments

Please amend the paragraphs [009] – [0011], [0013] – [0015], [0027], [0033] – [0034], [0044], [0046] – [0054], [0065] – [0067] and [0069] of the specification as follows:

[0009] FIGs. 1 and 2 are ~~is~~ a generalized block diagram of an exemplary computer system in which an example embodiment of the invention may be practiced.

[0010] FIG. ~~2~~ 3 illustrates a prior art method of changing individual bits in a control register.

[0011] FIG. ~~3~~ 5 illustrates a method of writing individual bits to a control register according to an example embodiment of the invention.

[0013] FIG. ~~5~~ 6 illustrates a read/write command setup protocol method which may be used in conjunction with a method of writing individual bits to a control register according to an example embodiment of the invention.

[0014] FIG. ~~6~~ 7 illustrates a conventional input/output (I/O) task-file access.

[0015] FIG. ~~7~~ 8 illustrate a streamlining task-file access which may be used in

conjunction with a method of writing individual bits to a control register according to an example embodiment of the invention.

[0027] ~~Although not~~ As shown in FIG. ~~1~~ 2, ICH 110 contains a plurality of controllers for the supported devices connected thereto. Exemplary supported I/O devices and peripherals include keyboards, input mouses, printers, scanners, display devices, hard disk drives, Compact Disk Read Only Memory (CD-ROM) drives, Compact Disk Read/Write (CD-RW) drives, and other types of storage devices. These controllers act as a communications translator between the supported devices and processor subsystem 101. They may include logic that runs protocol instructions out onto the bus connecting ICH 110 to the device. One of these controllers is an IDE interface controller or a controller compatible (including being backward compatible) with the IDE interface. One of these devices is a storage device that may require translation of processor instructions and may employ information stored in a location that may be connected with desktop computer system 100. It may be a disk drive that may be adapted to read and write at least one rigid magnetic data storage disk (hard disk) that rotates about a central axle. Despite the particulars of this example embodiment of the invention involving data transfers between an IDE interface controller in ICH 110 and one or more IDE storage devices, the invention is not limited thereto and may be applied to data transfers between any type of controller and device connected to the controller.

[0033] FIG. ~~3~~ 4 illustrates an example in which bit location 3 of a register must be set to "1" and bit location 1 must be cleared to "0". In this example, the software provides for an 8-bit value having a "1" in the bit locations of bit enable field ~~301~~ 401 that correspond to bit locations 3 and 1 of the register and a "0" in the bit locations of bit enable field ~~301~~ 401 that correspond to bit locations 2 and 0 of the register. This enables bit locations 3 and 1 of the register to be overwritten. It also provides a "1" in the bit locations of data field 402 that correspond to bit locations 2 and 0 of the register. (Any value may be provided at the bit locations of data field ~~302~~ 402 that correspond to bit locations 2 and 0 of the register.) In the example shown in FIG. ~~3~~ 4, the bit locations of bit enable field ~~301~~ 401 and data field ~~302~~ 402 are in the same position as the bit locations of the register.

[0034] Hardware associated with the register receives a data packet containing the bit enable field and the data field ("1010.sub.--1.times.0.times." binary in the example of FIG. ~~3~~ 4) and overwrites the bit locations of the register for which the enable bit in the corresponding location of the bit enable field is set. The other bit locations of the register are left unchanged. In the example shown in FIG. ~~3~~ 4, bit locations 3 and 1 of the register are overwritten with the data in the corresponding bit locations of data field ~~302~~ 402, while bit locations 2 and 0 of the register retain their initial values.

[0044] In a conventional I/O-based method, the CPU spends an average of greater than 1.2 microseconds per access that runs to the IDE drive as shown in FIG. 3

~~2~~. Although the IDE 1/0 accesses are used to initiate all disk accesses regardless of IDE mode (i.e. PIO, DMA, UDMA), this streamlining feature must only be used for UDMA transfers. This allows the driver software to off-load CPU 104 in processor subsystem 101 earlier to perform other activities while waiting for the interrupt upon the completion of the transfer.

[0046] FIG. 6 ~~5~~ illustrates a read/write streamlined command setup method ~~500~~ 600. FIG. 7 ~~6~~ illustrates a timeline of the method of FIG. 6 ~~5~~. As is readily seen, in FIG. 7 ~~6~~, while the ATA channel is in an active state, such as writing drive select, the CPU 104 is blocked from performing other tasks.

[0047] Method ~~500~~ 600 may include similarities to conventional protocols used to write a task-file. However, it differs at least insofar as it involves writing to a memory mapped register queue rather than I/O mapped task-file registers.

[0048] Method ~~500~~ 600 may be implemented in software recorded in any readable medium which, when executed, causes computer 100, preferably processor subsystem 101, to perform method ~~500~~ 600. In one embodiment, method ~~500~~ 600 may be implemented through a distributed readable storage medium containing executable computer program instructions which, when executed, cause at least one of a client computer system and a server computer system to perform method ~~500~~ 600. Additionally, method ~~500~~ 600 may be implemented though a computer readable storage

medium containing executable computer program instructions which, when executed, cause a computer system 100 to perform method ~~500~~ 600 .

[0049] Method ~~500~~ 600 may begin at step ~~502~~ 602. At step ~~502~~ 602, method ~~500~~ 600 may address the storage device so as to command the attention of the storage device. Bit four of the device/head register field indicates the selected device (DEV). Thus, step ~~502~~ 602 may include placing the proper input at bit four (the write/drive select bit) of the device/head register field. This bit four information is always sent to the I/O task-file.

[0050] At step ~~504~~ 604, method ~~500~~ 600 may read an alt-status register of the storage device to determine whether the storage device is busy. If it is busy, then method 500 may return a "SRB_STATUS_BUSY" signal at step 506 since the small computer system interface (SCSI) request block (SRB) field would not be clear. From step ~~506~~ 606, method ~~500~~ 600 may return to step ~~504~~ 604.

[0051] Under normal operations, the storage device may not be busy at the first reading of its alt-status register. Thus, the read command of step ~~504~~ 604 may be sent to the I/O task-file. Alternatively, method ~~500~~ 600 may return to step ~~504~~ 604 up to 20,000 times. Here, the read command of step ~~504~~ 604 may be sent to the memory queue.

[0052] If the storage device is not busy, then method ~~500~~ 600 may continue to step ~~508~~ 608. At step ~~508~~ 608, method ~~500~~ 600 may determine whether the DMA engine of the storage device is active. If the BM engine of the storage device is active, then the BM engine may be turned off and the drive reset at step ~~510~~ 610. If the BM engine of the storage device is not active, then method ~~500~~ 600 may proceed to step ~~512~~ 612.

[0053] At step 512, method ~~500~~ 600 may calculate the block count and the program device. This may involve writing the block length to the memory queue of the sector count register field. Step ~~512~~ 612 is distinguished from conventional techniques in that, under conventional techniques, the block length is written to an I/O task-file whereas step ~~512~~ 612 includes writing the block length to the memory queue.

[0054] At step ~~514~~ 614, method ~~500~~ 600 may calculate the logical block address (LBA) and the program device. This may include at least one of writing the following registers to the memory queue: sector number, cylinder low, cylinder high, and device/head register. Step ~~514~~ 614 is distinguished from conventional techniques in that, under conventional techniques, these registers are written to an I/O task-file whereas step ~~514~~ 614 includes writing the registers to the memory queue.

[0065] At step ~~516~~ 616, method ~~500~~ 600 may include programming the DMA descriptor table contents. At step ~~518~~ 618, the command register may be programmed

with a read or write command that may be sent to the memory queue rather than an I/O task-file.

[0066] At step ~~520~~ 620, the BM engine may be programmed. This may involve clearing the BM interrupt (BMI) status bit for a specifically-accessed controller, such as a controller in ICH 110. Additionally, the drive transfer protocol (DTP) of the BM engine may be set. In one embodiment, the BMI_DTP register may be set only once. Last, the BMI control may be set to a "Start/Stop Bus Master" bit.

[0067] With the BM engine programmed at step ~~520~~ 620, method ~~500~~ 600 may wait for an interrupt signal from the storage device at step ~~522~~ 622. This may involve returning a "SRB_STATUS_PENDING" signal since the small computer system interface (SCSI) request block (SRB) field would be connected with change. At step ~~524~~ 624, an interrupt signal may be received.

[0069] FIG. ~~7~~ 8 illustrates a timeline for a method of the invention wherein task-file access is streamlined. By using this example embodiment of the invention, the CPU is freed up to allow processing of other tasks.